

## WES-CS GROUP MEETING #3

### Exercise 1: Syntax

For this exercise, divide into groups of 2 or 3. Each group will get one set of cards: green, yellow, and pink, or buff, blue, and gold. The green/buff ones have an English description of some kind of Java code, the yellow/blue ones have a definition of the syntax for that code, and the pink/gold ones have an example of that kind of code. Your job is to match them up: find groups of three cards (one of each color) that go together.

### Exercise 2: Algorithms

Last week we assumed that we had a *Word* class that included a *swap* method:

```
void swap( int pos1, int pos2 )
```

Swap the letter in position *pos1* with the letter in position *pos2*.

and you wrote an algorithm (like the one given below) for reversing the letters in a word.

Step 1: Let *j* equal 0 and let *k* equal the number of letters in the word – 1.

Step 2: If *j* is greater than or equal to *k*, stop.

Step 3: `word.swap(j, k)`

Step 4: Let *j* equal *j*+1 and let *k* equal *k* – 1

Step 5: Goto Step 2.

This time, assume that the *Word* class also has methods that tell you

- what the *j*<sup>th</sup> character is
- the position of the *first* occurrence of a given letter in the word
- the position of the *last* occurrence of a given letter in the word

Write an algorithm to transform an arbitrary *Word* called *original* into the anagram in another *Word* called *anagram* (assume that the two words really do have the same letters, just in a different order). You don't have to write actual code, just say things like “*let j be the number of characters in original*”, “*let ch be the character at position j in the original word*” etc.

Write your algorithm as a sequence of steps like the anagram given above. Make sure that you always eventually get to a “stop” step, so you know when you're done.

## Exercise 3: The Car Class

This exercise will help you to understand what happens when objects are declared and created, and when methods are called.

It will also help you to understand the difference between copying from one variable to another when the variable is an object, and when it is a primitive type (*int*, *double*, *boolean*, etc), as well as the difference between changing the values of variables that are objects vs primitive types.

First, take a look at the *Car* class defined on the next page.

Now execute the following code fragment; let one person play the role of each variable (*myCar*, *yourCar*, *anotherCar*, *oldSpeed*, and *currSpeed*). When a variable is assigned to, or one of its methods is called, the person playing the role of that variable should act out effects of the assignment or call.

```
Car myCar, yourCar, anotherCar;
int oldSpeed, currSpeed;

myCar = new Car("beep");
yourCar = new Car("honk");
anotherCar = myCar;

currSpeed = myCar.getCurrSpeed();
yourCar.changeSpeed(7);
anotherCar.changeSpeed(20);
currSpeed = myCar.getCurrSpeed();

myCar.blowHorn(2);
yourCar.blowHorn(3);
anotherCar.blowHorn(4);

oldSpeed = currSpeed;
myCar = yourCar;
currSpeed = myCar.getCurrSpeed();

myCar.changeSound("ooga");
myCar.blowHorn(currSpeed/5);
yourCar.blowHorn( yourCar.getCurrSpeed()/2 );
anotherCar.blowHorn( myCar.getCurrSpeed()/10 );
anotherCar = myCar;
```

```

class Car {
    /*******
     * data members
     *****/
    private int currSpeed;
    private String hornSound;

    /*******
     * public methods
     *****/
    /* constructor */
    public Car(String sound) {
        currSpeed = 0;
        hornSound = sound;
    }

    /* changeSound: change the horn sound */
    public void changeSound(String newSound) {
        hornSound = newSound;
    }

    /* blowHorn: blow the horn;
     * parameter numTimes tells you how many times
     */
    public void blowHorn(int numTimes) {
        while (numTimes > 0) {
            System.out.println(hornSound);
            numTimes--;
        }
    }

    /* changeSpeed: change speed */
    public void changeSpeed(int milesPerHour) {
        currSpeed = currSpeed + milesPerHour;
    }

    /* getCurrSpeed: return the current speed */
    public int getCurrSpeed() {
        return currSpeed;
    }
}

```

## Exercise 4: Java variables and types

Remember that in Java

- variables must be declared and initialized before they are used,
- operators (like +, -, and so on) must be applied only to variables and literals of the correct type,
- in general, the types of the left and right-hand sides of an assignment must match (but note that an *int* value can be assigned to a *double* variable, but not vice-versa),
- the types of the arguments in a method call must match the types of the corresponding parameters, and the value returned must match the method's return type (again, an *int* can be used where a *double* is expected, but not vice-versa).

Keeping all this in mind, find all of the errors that the Java compiler would report in the following code.

```
public int doDivision( double denom ) {
    int returnVal;
    double num1 = 4.4;
    num2 = 5.5;
    returnVal = (num1 + num2)/denom;
    return returnVal;
}
```

```
public int computeVal( int oneVal ) {
    int returnVal;
    int i1, i2, i3;
    Integer intObj;
    i1 = 222222222222222;
    intObj = 11;
    i2 = num1;
    i3 = doDivision( "hello" );
    returnVal = (i1 + i2)/10000.0;
    Return returnVal;
}
```

## Exercise 5: Logical Thinking

Assume that you have 8 coins, and you know that 7 are OK but one is bad. You know that the bad coin has a different weight than the good coins, but you don't know whether it's heavier or lighter.

Figure out how, using only a balance scale, you can find out which is the bad coin using just 3 weighings. Hint: Find a way to determine that half of the coins are OK with just 1 weighing.

Now figure out which is the bad coin assuming that you have *nine* coins, one of which is bad. (Still use just 3 weighings to find the bad coin.)

And now for a real challenge, do the same thing assuming that you have 13 coins.

## ***GREEN OR BUFF CARDS***

variable declaration (no initialization)

variable declaration with initialization

object declaration (no object creation)

object declaration and creation

class declaration

(non-constructor) method declaration

constructor declaration

method call

## ***GREEN OR BUFF CARDS***

assignment statement

field (instance variable, data member) declaration

print to the terminal

comment

## ***YELLOW OR BLUE CARDS***

<type> <var name>;

<type> <var name> = <expression>;

<class name> <object name>;

<class name> <object name> =  
new <class name> ( <args> );

<access specifier> class <class name> {  
    <class member declarations>  
}

<access specifier> <return type> <method name>( <params> ) {  
    <method body>  
}

## ***YELLOW OR BLUE CARDS***

```
<access specifier> <class name>( <params> ) {  
    <method body>  
}
```

```
<object name>.<method name>( <args> );
```

```
<var name> = <expression>;
```

```
<access specifier> <type> <name> ;
```

```
System.out.print( <exp> );    or    System.out.println(<exp>);
```

```
// <text>           or           /* <text> */
```

## *PINK OR GOLD CARDS*

```
int x;
```

```
Artist picasso;
```

```
double d = 5.5;
```

```
String name = "Bozo";      or      Time t = new Time(12, 10);
```

```
public class Time {  
    data member declarations  
    method declarations  
}
```

```
/* this is the tricky part */      or      // constructor
```

```
public void drawLineLeft( int len ) {  
    statements  
}
```

## ***PINK OR GOLD CARDS***

```
public Time( int hrs, int mins ) {  
    statements  
}
```

```
picasso.drawLineLeft( 10 );
```

```
x = y + 5;
```

```
System.out.print("hello");    or    System.out.println(x);
```

```
private int hours;
```